AFRL-IF-RS-TR-2005-296
Final Technical Report
August 2005

OPTIMIZED PARALLEL DISCRETE EVENT SIMULATION (PDES) FOR HIGH PERFORMANCE COMPUTING (HPC) CLUSTERS

SUNY at Binghamton

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2005-296 has been reviewed and is approved for publication.

APPROVED: /s/

JAMES P. HANNA
Project Engineer

FOR THE DIRECTOR: /s/

JAMES A. COLLINS, Acting Chief
Advanced Computing Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	AUGUST 2005	Final Feb 04 – Jan 05

4. TITLE AND SUBTITLE
OPTIMIZED PARALLEL DISCRETE EVENT SIMULATION (PDES) FOR HIGH PERFORMANCE COMPUTING (HPC) CLUSTERS

5. FUNDING NUMBERS
C - FA8750-04-1-0054
PE -
PR - PDES
TA - HP
WU - C4

6. AUTHOR(S)
Nael Abu-Ghazaleh

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
SUNY at Binghamton
4400 Vestal Parkway East
Binghamton New York 13902-6000

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory/IFTC
26 Electronic Parkway
Rome New York 13441-4514

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2005-296

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: James P. Hanna/IFTC/(315) 330-3473/ James. Hanna@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***

The aim of this project was to study the communication subsystem performance of state of the art optimistic simulator Synchronous Parallel Environment for Emulation and Discrete-Event Simulation (SPEEDES) and explore approaches for accelerating it. The primary focus has been on system level improvements; specifically, a primary goal was to explore the performance of the simulator on a cluster with a Myrinet network. Myrinet provides several features conducive to the success of parallel simulation: it provides a high bandwidth, low latency network, with user level network protocol (avoiding the Operating System overhead) and provides some processing on the network interface card to reduce the load on the host processor. In addition, algorithmic improvements to the communication susbsystem were considered.

14. SUBJECT TERMS
Myrinet, SPEEDES, Symmetric Multiprocessors, Simulation Engine, High Performance Network, Parallel Discrete Event Simulation, Sockets-GM

15. NUMBER OF PAGES
16

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

TABLE OF CONTENTS

LIST OF FIGURES

1. Introduction

Simulation is a critical capability at the heart of several Air Force and DoD applications; for example, it is used for model analysis, war-gaming, and complex system design and analysis. Increasing the performance and the capacity of simulation has considerable implications on the success of these applications – more complex scenarios can be analyzed in more detail to provide more accurate results in a shorter time. Parallel Discrete Event Simulation (PDES), which leverages the power of parallel processing, can significantly improve the performance and capacity of simulation. However, PDES falls short on delivering such performance because of its fine-grained and dynamic nature due to the complex dependencies between the simulation processes. This is especially true in cluster environments where the cost of communication is more expensive than Symmetric Multiprocessors or custom parallel machines. Since clusters provide an overwhelming cost-to-performance solution point, developing effective solutions to PDES and other fine-grained/dynamic applications for them will provide a valuable capability to the Air Force and significantly improve its simulation capabilities.

Fine grained and dynamic applications such as Parallel Discrete Event Simulation [Fujimoto 1990, Metron] present a challenge to clusters. Their fine grained nature makes simulation communication bounds for many models; since the cost of communication is significantly higher than computation; frequent communication limits the performance and scalability of fine-grained applications. The existing implementation of SPEEDES[1] uses a centralized server; all communication occurs first to this server and then to the destination. While this organization has several desirable qualities (such as simplifying external interfacing to the simulation, and providing a logical point for implementing group communication operations), it significantly increases the communication cost (already a major bottleneck), and reduces the scalability of the simulation. In previous work, we had implemented an all to all version of this library that allows many direct event messages (but not group communication messages) to be exchanged directly between the simulation nodes. However, significant opportunities remain to improving this critical portion of the simulation engine. Exploring this space is a long term goal of our research.

In this project, we attempted to address the communication latency problem by taking advantage of the Myrinet high performance network from Myricom [Myrinet]. At the onset of the project, none of the available communication libraries for using Myrinet provided acceptable performance. In particular, the GM-sockets library which implements Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) sockets abstraction for Myrinet provided surprisingly poor performance. Using the native sockets library on top of GM (the inefficient existing operating system

[1] SPEEDES is a state-of-the-art parallel simulator from Metron, Inc [Metron]. It is based on Jeff Steinman's work at Caltech, and employs an innovative synchronization model (Breathing Time Warp) and checkpointing model [Steinman 1993]. It is in use in several DoD, and especially Air Force, projects.

sockets implementation is used, requiring operating system mode switch and several buffer copies) yielded only small improvements over the regular sockets over Gigabit Ethernet implementation. Finally, the Message Passing Library (MPI), which has an implementation for Myrinet was a possible avenue. However, Metron had stopped supporting the MPI version of SPEEDES and the old distribution was no longer operational. Thus, no options existed for taking advantage of the Myrinet network for simulation. Providing such an option was a primary goal of this project.

The remainder of this report is organized as follows. Section 2 describes Parallel Discrete Event Simulation and the SPEEDES simulator in more detail. It also describes the target cluster used in the experiments. Section 3 describes the development and implementation effort undertaken throughout this project. Section 4 presents a performance study of the SPEEDES simulator with MPI for Ethernet and Myrinet using a Benchmark from the Distributed Information Enterprise Modeling and Simulation (DIEMS) Joint Battlespace Infosphere (JBI) project. Finally, Section 5 presents some concluding remarks.

2. Background and Related Work

In this section we first overview PDES, and the motivation for more effective fine-grained communication and self-monitoring and adaptation. We then describe the HHPC environment in more detail.

2.1 Parallel Discrete Event Simulation

In Discrete Event Simulation (DES), a model starts with an initial state and an initial number of scheduled events. Events are ordered by simulation time in an event queue. Simulation proceeds by processing the earliest event, which can cause changes in the simulation state and schedule for one or more future events. The simulation time advances to the time of the earliest unprocessed event. Simulation terminates when there are no more events to process or when a predetermined simulation time is reached.

Parallel Discrete Event Simulation (PDES) leverages parallel processing to attempt to accelerate the performance and capacity of DES. The simulation model is partitioned across multiple simulation processes (called Logical Processes, or LPs). Each LP maintains a local event queue and carries out simulation as described above (repeatedly processing the earliest time stamp event). A locally processed event may generate events to remote LPs (that is, these events may cause changes to the state managed by the remote LP). Thus, LPs communicate by exchanging time-stamped event messages [Jefferson-83]. Correct simulation requires that all events be processed in time-stamp order. Therefore, a synchronization model is needed to ensure that remote events are processed in time-stamp order.

Conservative PDES simulators carry out synchronization as follows: an LP, LP_i, does not process an event until it is guaranteed that no other LP will generate a remote event destined to LP_i with a time stamp earlier than the current earliest event. Thus, explicit time step synchronization of the LPs is needed, severely limiting the potential event processing concurrency. Alternatively, optimistic PDES simulation (the so-called Time-Warp model [Fujimoto-90]) does not require explicit synchronization among LPs. Each LP enforces causal ordering on local events (by processing them in time-stamp order). Causality is preserved on remote events by detecting causality errors (when a *straggler* event with a time-stamp in the past is received) and recovering from them by rolling-back the simulation to a state prior to the time of the straggler event. Thus, each simulator must maintain state and event histories in order to enable recovery from straggler events. This state information must be garbage collected to control memory usage and enhance the simulator locality; a Global Virtual Time (GVT) algorithm is used to detect the global progress time of the simulation, allowing the garbage collection of histories earlier than this time. We use the SPEEDES simulation environment [Metron] in our studies; SPEEDES is a state of the art PDES simulator that uses Breathing Time Warp (an optimistic simulator that bounds optimism to attempt to control excessive rollbacks) [Steinman 1993]. SPEEDES is used in several projects in the Air Force and DoD.

The performance of PDES is heavily influenced by the message exchange latency: PDES is fine-grained, with event messages generated frequently (depending on the model) to remote LPs. Delays in receiving these messages can cause the simulation to be erroneous at remote LPs (since the event message will be received after the simulator has moved past it). It has been shown that improving the performance of the communication subsystem results in significant improvement in the simulation performance (e.g., [Chetlur98, Sharma99, AbuGhazaleh04]). Furthermore, slow message exchange causes GVT estimates to be slow, limiting the available concurrency in Breathing Time Warp and reducing the efficiency of garbage collection (increasing the simulator memory footprint and worsening the average memory access time).

In addition, the simulation behavior is dynamic and unpredictable. There are several parameters that configure the simulation and the sub-algorithms used in it (for example, several parameters are used in Breathing Time Warp to control the degree of tolerated optimism); a suitable configuration depends on the current model behavior and can have a large effect on the simulation performance. Moreover, the model partitioning affects the resulting remote dependencies and can also significantly influence performance. Furthermore, the dependencies in the model can evolve dynamically (for example, as objects move away from initial close objects and closer to other objects). Thus, effective configuration and partitioning is necessary both initially and dynamically during run time (although this is beyond the scope of this project).

2.2 Heterogeneous High Performance Computing (HHPC) Cluster

HHPC is a Beowulf cluster made of commercial off the shelf personal computers (featuring dual processor Xeon's) interconnected via a Gigabit Ethernet Network and a Myrinet network [Boden 1995]. In addition, each node has an Annapolis Micro Devices

(AMD) Wildstar II FPGA board on the Peripheral Component Interconnect (PCI) bus. The Wildstar has a Xilinx Virtex II FPGA, some DRAM and SRAM banks and an LVDS I/O card. We use the I/O card to interconnect the Field Programmable Gate-Arrays (FPGAs) directly to each other using a custom built all-to-all serial board. This board provides connectivity from every node to every other node concurrently using a dedicated serial line. This results in a low-latency but low-bandwidth communication channel among the FPGAs. Without this connectivity, all communication must go through the communication fabric at a latency ranging from around 10 microseconds (for the expensive Myrinet) to several 10s of microseconds for the commodity Gigabit Ethernet.

3. Communication Subsystem Development Effort

A major component of the development effort went to attempting to address the performance problems observed in the GM-sockets library. Unfortunately, this effort was not successful – the available library was unstable and bug ridden (functional as well as performance bugs). Several features in the library are not implemented. Finally, Myricom has stopped supporting this library and moved to alternative libraries that are not compatible with the version of the network hardware we have available to us. We detail these efforts in Subsection 3.2. The other component of the effort targeted the MPI library support for SPEEDES. This effort was successful (described in Subsection 3.1), and led to a version of the simulator that uses the Myrinet network. Moreover, the MPI implementation for Ethernet is a possibly useful byproduct of this effort.

3.1. SPEEDES-MPI Development Effort

Earlier versions of SPEEDES used MPI. However, Metron developed a sockets implementation and decided to stop supporting the MPI implementation. Efforts to discuss this implementation with them were unfruitful. The distributed code from an old version of SPEEDES was not operational, causing the simulation to fail or to freeze. The effort in this component of the work consisted of studying and understanding the old MPI communication library, interfacing it with the current version of SPEEDES and debugging the problems preventing it from operating correctly. We discovered and corrected several problems in this process. Some problems were due to the fact that the SPEEDES code changed, and no longer interfaced correctly with this old implementation. In addition, we discovered several significant errors in the implementation. For example, one error caused some messages to be consumed by the library (without getting delivered to the simulator). This caused some event messages to be lost and led to erroneous simulation (and simulation failures sometimes), as well as deadlock if the lost message was part of a blocking operation. Another error caused partial receive of some messages, leading to immediate failure of the simulation.

A limitation of the implementation (only with Ethernet) is that the mpirun script fails to pass the working directory correctly to SPEEDES. This results in the simulation starting in the top directory on other machines. Thus, to collect the results, I had to run from the

top directory. Because the use of MPI was limited to go through the SUN grid engine and the system-wide installation of MPI, I was not able to investigate changes to the script.

Another minor limitation of both versions is that the MPI_Finalize exits the processes sometimes before the remote stdio files have finished writing (if extremely large volume outputs are generated at the end). This problem was faced only with the FixedSpeedup_64D benchmark which writes over 1 Megabytes of data as it is exiting. This problem can be worked around either by delaying the call to MPI_Finalize, or by writing the output more incrementally or to files.

3.2. GM/sockets development Effort

We attempted several approaches to reach a working and efficient implementation of sockets over GM. Our initial experiments with the GM-sockets library showed extremely poor performance (lower than that of regular sockets over GM, and even poorer than sockets over Ethernet). In addition, on standard micro-benchmarks such as netperf, several benchmarks failed (crashing, or getting stuck indefinitely). Finally, there were significant problems in getting SPEEDES to compile, and eventually, run correctly with this library.

The progression of our efforts is as follows. At first, we attempted to fix the problems with the GM-sockets library; because of the effort required to develop a communication library from scratch, this appeared as the path of least resistance. Furthermore, we were encouraged because the library (some versions) is implemented completely in user space, making development easier. This part of the work took a significant amount of effort; we discovered several bugs and performance bottlenecks then reported them to Myrinet. They moved to a new version during development, and then had further releases based on the problems we discovered. Finally, they recommended that we move to their new libraries which require GM-2 and/or the MX libraries, and are not compatible with our older generation hardware. As a result, we attempted to bypass the GM-sockets library completely by interfacing SPEEDES with GM. However, after implementing a basic subset of the communication calls, we found that we required reimplementation of most of the sockets library functionality which was a larger task than our available resources. In the following section, we report on our efforts in more detail, and present an evaluation result with simple microbenchmarks of the different approaches we attempted.

3.2.1. Experiences in Using and Debugging the GM-Sockets library

GM-sockets is a user level library for implementing sockets on top of the GM message library (the low-level message passing abstraction for accessing the Myrinet network). Theoretically, it should provide significant improvements over regular sockets on top of GM (where regular TCP/IP processing occurs, and then the myrinet device is accessed as an Ethernet device via an Ethernet emulation layer). The advantages occur for the

following reasons. Much of the sockets library functionality can be discarded or simplified (e.g., congestion control and checksums) because the underlying Myrinet network handles this functionality in hardware. Furthermore, buffer copies can be reduced because the library operates from user space, and the packet can be dispatched directly from user space to the device. Finally, the overhead of switching over to the O/S is avoided. As a result, latencies in the order of 10 microseconds can be expected, compared to 100 microseconds or more for Ethernet. Another advantage occurs because some of the message processing occurs on the LANAI card, freeing the host processor to focus on its primary computation.

GM-sockets had three different ways of mapping the socket calls to raw GM calls: the kernel module, the dynamic user level library, and the static user level library implementations. The dynamic user level library implementation did not work from the start; we were advised at that point to use the static library instead. According to the functionality of this library, it was supposed to provide a seamless integration of the socket calls with the underlying GM. However, the examples given showed no performance improvement using this library (in fact, performance degradation resulted in many cases). Using the netperf microbenchmark suite, the results were worse than regular sockets over Ethernet in terms of bandwidth and latency; moreover, several examples simply failed.

Sockets-GM 1.2.1 was the latest version of the library on Sep 12th 2004. When SPEEDES was integrated with the Sockets-GM, it was found that the program used to wait indefinitely. By going through the SPEEDES code, it was found that the application used to abruptly hang during the sendmsg() (a UDP sending API) call . It was decided
to test Socket-GM with simple examples using sendmsg(). The simple examples failed to work. At this stage we started going through the Socket-GM library code. It was found that the library was trapping some of the socket calls, but it failed to recognize the sendmsg() call. Upon mailing to the Myrinet support, they advised us to use the newer version, Sockets-GM 1.3. The Sockets-GM 1.2.1 was an incomplete version of the wrapper library and did not support all the calls, which failed to run an application like SPEEDES which exercises many features of the sockets library.

The use of Sockets-GM 1.3 on SPEEDES led to triggering of the SIGSEGV signals. The hostnames were not being correctly recognized by the library. Integration of SPEEDES with the latest version of Sockets-GM needed changes to the system configurations. According to the new specifications, the hostnames of all the machines had to be changed to reflect a new format. It was now required to have the IP address followed by the board number. After all these changes, we were back on square one when the SPEEDES was still hanging at sendmsg() call. Later, testing simple examples and adding debug statements in the
Sockets-GM 1.3 verified that sendmsg() was not being recognized by Sockets-GM 1.3 as in the previous version. The original sendmsg() was being called instead of the wrapper function present in the library, which obviously led to a send error. The receiving application used to wait till the sender sent the data and hence the application used to wait

infinitely. The Myrinet support was again consulted and they advised us to check the compile time warnings of Sockets-GM 1.3 which was later verified to be absent.

Debugging the symbols from the library, it was later found that there was no sendmsg() function in the text section. However, other simple functions like send() were present in the text section of the library. This meant that the sendmsg() function that was called was never present in the library. There was a minor signature mismatch between the linux sendmsg() and the one declared in the Socket-GM. After altering the code of Sockets-GM 1.3 to match the signature of the sendmsg in the linux to include file socket.h, the sendmsg was called but the server used to terminate with a segmentation fault. This bug was again transmitted to the Myrinet support team and after a couple of days, the code was fixed and source for the newer code (Sockets-GM-ULEVEL-1.3+1) was provided.

This new release fixed the above problem but a new problem surfaced. The SPEEDES application was now waiting forever in the some of the recv() function calls. It was found that the recv() used to hang whenever the MSG_PEEK flag was set and when the receive was non-blocking. Upon a non-blocking recv() call with MSG_PEEK flag set, the call should return a -1 with errno set to ``E AGAIN''. But the Socket-GM used to block till there was data available on the socket. The SPEEDES application used to check the presence of data with non-blocking receives with MSG_PEEK set. This resulted in an undesirable infinite wait time. The Myrinet support team was contacted again and they indicated that this feature is only available in the module version of the Sockets-GM. The module version could not be verified because changes to the system at the root level were needed.

The kernel module implementation needed changes to be done at the system level by the root. Furthermore, it required the GM-2 library to be installed in place of the GM library we had on the development cluster. However, the GM-2 library is not recommended for the version of Myrinet hardware that we have available on the HHPC machines. They also informed us that they are not actively maintaining the GM version anymore, and recommended that we move to either GM-2 or the newer MX library (which also does not support our hardware).

3.2.2. Integrating SPEEDES with RAW GM

To avoid using the GM-sockets library, we investigated porting SPEEDES to directly use the GM message layer. This effort required changes in the communication API of SPEEDES from using sockets to using GM calls. The available options for this conversion were:

1. To change all the SPEEDES socket calls to raw GM calls.
2. To add wrapper functions that transparently convert all socket calls to the GM calls.

Handling all the cases using the first approach required major changes and verification of the SPEEDES code. Moreover, several aspects required a major rewrite of the SPEEDES

communication support; specifically, raw GM does not implement reliable communication. We elected to pursue the second approach.

The periodic issues with Sockets-GM and the regular version changes were hard to track and debug. A thin layer of the socket-to-gm mapping API, henceforth called as *sock2gm*, was written. The plan was to replace the existing socket API calls in SPEEDES with the new API send and receive calls. A raw GM send and receive was written to handle simple send and receive.

This paragraph describes the brief design of *sock2gm*. Instead of using socket descriptors, the sock2gm wrapper functions assign a unique number (called *sock2gm ID*) to each of the sock2gm sockets opened. The sock2gm ID is mapped to a GM port opened by the sock2gm layer. Sending and receiving can be mapped by specifying the sock2gm ID obtained by the opening of the sock2gm socket. The function calls were similar to the raw GM send and receive calls, but the necessity to keep track of the callbacks and waiting for events are hidden by the sock2gm API. Simple experiments with connectionless message passing were tested and verified. The below list describes some of the important functions of sock2gm API.

1. sock2gm_init(): This is the initialization function that initializes GM by calling gm_init(). This method has to be called before any sock2gm API calls. It also initializes the data structures needed to maintain the mapping of sock2gm ID to GM port.
2. sock2gm_open(): This method is analogous to the socket() call. It opens a GM port and assigns a unique sock2gm ID and returns the ID to the caller. It enables remote memory access to the port and sets up GM priority to the port.
3. sock2gm_alloc(): This allocates the DMA memory to a given sock2gm ID that can be used by send and receive functions.
4. sock2gm_send(): Sending the data to a target node can be invoked by calling this function. It resolves the mapping of sock2gm ID to the respective port and will send the data with callback to the target node. The function is blocking and will wait till the send completion event is triggered.
5. sock2gm_receive(): This function is the counterpart for socket receive function. It is blocking in nature and waits till the arrival of data on the requested GM port.

The work is ongoing and the future plans are to:

1. Enable sock2gm API for the connection oriented calls.
2. Enable advanced options like peeking into the buffer for checking the presence of message, blocking and non-blocking send and receive.
3. The current sock2gm API hides the callback overhead from the sender and receiver. It would be easier to replace all the socket calls with sock2gm calls if the API matches the currently used socket API.
4. The current mapping of sock2gm ID to a GM port is maintained in a linked list structure. It would be more efficient to use a hash table data structure.

The current implementation maps one sock2gm ID to a port. Mapping multiple IDs to a port is also desirable.

4. Performance Evaluation with MPI on Ethernet and Myrinet

The performance of the FixedSpeedup_64D benchmark using the different communication alternatives (sockets/Ethernet; MPI/Ethernet and MPI/GM) is shown in Figure 1. The benchmark represents a JBI scenario modeled using the DIEMS JBI modeling and simulation environment which is layered on top of SPEEDES. We used the best performing configuration we found for the sockets implementation, Breathing Time Warp with (Ngvt=1000, Nrisk=500, Nopt=1000), against a typical configuration for the MPI versions with (Breathing Time Warp with Ngvt=2000, Nrisk=4000, and Nopt=8000). We simulated other configurations: the two MPI versions appeared most stable to changes in the configuration and could tolerate aggressive GVT computation. The same did not hold true for the sockets version whose performance degraded quickly both for high optimism cases (the extra delay causing additional rollbacks), as well as when optimism was bounded closer (due to the extra delay required by the additional GVT computations).

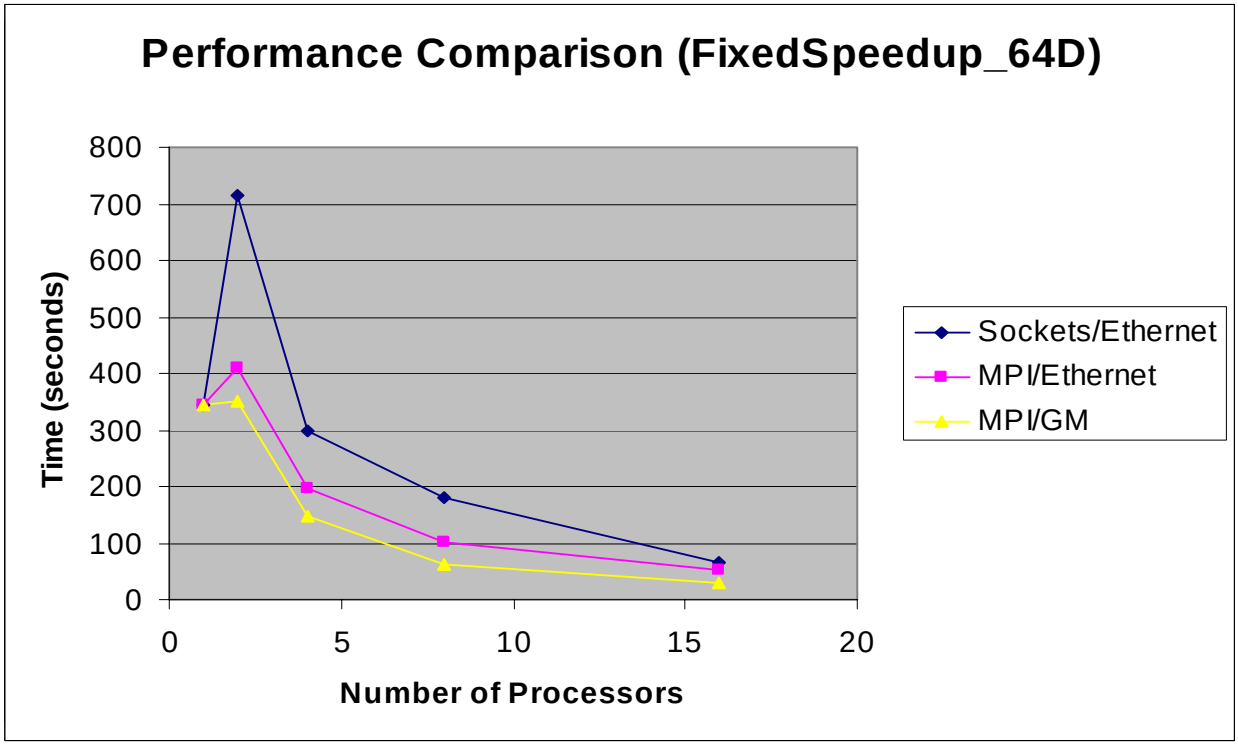

Figure 1: The performance of the FixedSpeedup_64D Benchmark

In this base implementation, the MPI version significantly outperforms the sockets version on Ethernet (for example, requiring nearly half the time for 4 processor and 8 processor configurations). Likewise, the MPI GM version was considerably faster than

the MPI Ethernet version (by a factor of 40% to 100%). As a result, the MPI GM version was between 2 to 4 times faster than the base sockets over the Ethernet version for this benchmark.

In the second study, we evaluated the effect of increasing the impact of communication latency on the performance of MPI with and without GM. In this study, we kept the model the same, but increased the frequency of carrying out GVT by reducing the Ngvt, and Nrisk parameters. Thus, the faster the GVT computation, the faster can the whole simulation advance.

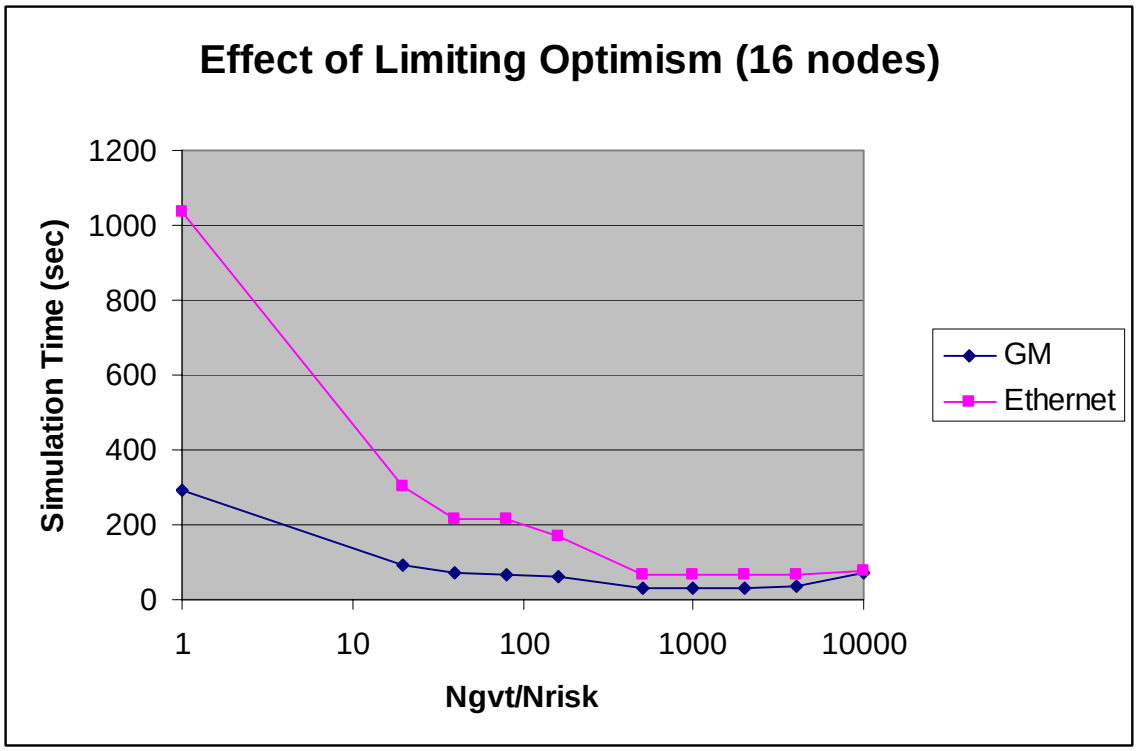

Figure 2: Effect of limiting optimism on the two MPI implementations

The results for this study are shown in Figure 2. In this figure, the frequency of GVT computation, as well as the number of optimistically processed events (both Nrisk and Nopt), was varied in lockstep (all were changed to the same value). In general, depending on the model and the communication latency, there is a level of optimism that will yield effective performance (ignoring load balancing issues). At this level, the optimistic computation is most often correct. If optimism is increased significantly beyond this point, one would expect the performance to suffer as more events are likely to be processed erroneously. In our model, the application is quite tolerant to optimism (it has high look-ahead, meaning that the remotely generated events occur far enough in the future to allow time for them to be received). In other models that have less overhead, the simulation may not tolerate high optimism and will require more frequent GVT computation to increase its efficiency.

To study the effect of communication latency and to simulate applications that do not have high look ahead, we varied the degree of optimism for the two MPI implementations. The primary observation here is that the GM version could gracefully tolerate really aggressive GVT computation (up to once every 15 events) with little degradation in performance. Conversely, the Ethernet version starts to slow down significantly at GVT period of a few hundred events. This bodes well for other applications that do not have the relaxed look ahead requirement as does this one.

A final observed advantage for the MPICH-GM version is the amount of time spent in startup (this time is not included in the analysis above). Because the Ethernet network is a shared medium that is also used to connect the machines to the shared filesystem, the communication traffic has to contend with the filesystem traffic for access to the network. This effect is most noticeable at the simulation startup when the different nodes have to copy over the large executables through the Ethernet interface, essentially one at a time. In contrast, the myrinet network allows concurrent pipelined sends of these executables and results in a noticeably faster startup time for the simulation (less than one second, compared to 10+ seconds for Ethernet in a 16 node run).

5. Concluding Remarks

The primary goal of this work was to develop a communication subsystem for SPEEDES that enables it to capitalize on the performance opportunity afforded by the Myrinet network infrastructure. While our initial plan to carry out this project by addressing the performance problems in the GM-sockets library, for many reasons we were not able to reach the goal via that path. Instead, we upgraded and debugged existing MPI support that was not functional and was no longer supported by Metron. With a working MPI version, we were able to use the Myrinet network via the mature MPICH-GM library available for it. This led to a working prototype, that shows significant performance advantages on the benchmark that we studied.

A lesson learned from this effort is that the GM-sockets library for past generation GM-1 based Myrinet hardware is not fully compliant with the sockets library and suffer from functional and performance issues. Moreover, it appears that the vendors have stopped attempting to address these limitations and have focused their efforts (understandably) on support for their newer lines of products. Unfortunately, this has led to our inability to generate a working prototype with the incomplete and buggy libraries available for our legacy 3-4 year old system.

We did not present the low level microbenchmark evaluation of this library (we have these results); surprisingly, these results do not show a dramatic advantage for Myrinet over Gigabit Ethernet (while this could be expected in terms of bandwidth, it appears that the gap is closing in terms of latency as well). Similar results were observed at the application level by other researchers (e.g., Myrinet was found to achieve only 5% performance improvements on the NAS benchmark [Majumder 2004]). With 10-Gig Ethernet becoming available, its unclear whether custom built networks such as Myrinet will continue to play a big role in the cluster computing arena.

Finally, a personal lesson learned is that it is difficult to pull students to a research area quite different from theirs and expect that they will be productive immediately. It took me a good 6 months time to train the students to the point where they became truly capable of conducting the work and able to work their way past difficult problems. In a project whose lifetime was 1 year, this had a major impact on the outcome of our GM-sockets effort.

References

[Fujimoto 1990] R. Fujimoto, "Parallel Discrete Event Simulation", Communications of the ACM, 33(10):30-53, Oct. 1990.

[Metron] Metron, Inc., "The SPEEDES API Reference Manual", 2002 (available from http://www.SPEEDES.com)

[Jefferson 1983] D. Jefferson, "Virtual Time", Proc. Of the International Conference on Parallel Processing, 1983, pages 384—394

[Steinman 1993] J.S. Steinman. "Breathing Time Warp". In 7th Workshop on Parallel and Distributed Simulation (PADS'93), pages 109--118, May 1993

[Chetlur 1998] http://doi.acm.org/10.1145/278008.278017M. Chetlur, N. B. Abu-Ghazaleh, R. Radhakrishnan, P. A. Wilsey: Optimizing Communication in Time-Warp Simulators. Workshop on Parallel and Distributed Simulation 1998: 64-71

[Sharma 1999] Girindra D. Sharma, Radharamanan Radhakrishnan, Umesh Kumar V. Rajasekaran, Nael B. Abu-Ghazaleh, Philip A. Wilsey: Time Warp Simulation on Clumps. Workshop on Parallel and Distributed Simulation 1999: 174-181

[Abu-Ghazaleh 2004] N. Abu-Ghazaleh, R. Linderman, R. Hillman and J.Hanna, "Using a Heterogeneous High Performance Cluster for Parallel Discrete Event Simulation", High Performance Computing User Group, 2004.

[Boden 1995] N. Boden, D. Cohen, R. Felderman, A.Kulawik, C. Seitz, J.Seizovic and W-K. Su, "Myrinet: A Gigabit-per-second Local Area Network Network", **IEEE Micro, 15(1):29--36, February 1995.**

[Majumder 2004] S. Majumder and S. Rixner, "Comparing Ethernet and Myrinet for MPI Communication," Proceedings of the Workshop on Languages, Compilers, and Run-Time Support for Scalable Systems (LCR) (2004).